Information Retrieval: Lecture Exercises & Solutions

Lecture 1: Introduction & Boolean Retrieval

Exercise 1 (Page 45)

Exercise: Adapt the merge for the queries: **Brutus AND NOT Caesar**, **Brutus OR NOT Caesar**. Can we still run through the merge in time $O(x + y)$?

Answer:

- **For 'Brutus AND NOT Caesar':** You can adapt the standard 'INTERSECT' merge algorithm to perform a set difference. Let p_1 be the postings list for 'Brutus' and p_2 be for 'Caesar'.

  1. Iterate through both lists.
  2. If $docID(p_1) == docID(p_2)$: This document contains both, so it's *not* a match. Advance both pointers.
  3. If $docID(p_1) < docID(p_2)$: This document contains 'Brutus' but *not* 'Caesar' (since p_2 is already past it). Add $docID(p_1)$ to the results. Advance p_1.
  4. If $docID(p_1) > docID(p_2)$: This document contains 'Caesar' but not 'Brutus'. Skip it. Advance p_2.
  5. When p_2 reaches the end (NIL), add all remaining docIDs from p_1 to the results.

  This operation still completes in $O(x + y)$ **time**.

- **For 'Brutus OR NOT Caesar':** This query is problematic for a simple merge. The set 'NOT Caesar' refers to *all documents in the entire collection* that do not contain 'Caesar'. This list would be massive, making a merge inefficient. This query would retrieve all documents that contain 'Brutus', *plus* all documents that do not contain 'Caesar'. This operation **cannot** be run in $O(x + y)$ time because the size of the 'NOT Caesar' list is not y; it is $N - y$ (where N is the total number of documents).

Exercise 2 (Page 49)

Exercise: Recommend a query processing order for **(tangerine OR trees) AND (marmalade OR skies) AND (kaleidoscope OR eyes)**. Which two terms should we process first?

Frequencies:

- eyes: 213,312

- kaleidoscope: 87,009
- marmalade: 107,913
- skies: 271,658
- tangerine: 46,653
- trees: 316,812

Answer: The approach should be to estimate the size of each 'OR' clause by summing their frequencies, and then process the 'AND' operations in increasing order of these estimated sizes.

1. **Estimate 'OR' clause sizes:**
   - (kaleidoscope OR eyes): $87,009 + 213,312 = \mathbf{300,321}$
   - (tangerine OR trees): $46,653 + 316,812 = \mathbf{363,465}$
   - (marmalade OR skies): $107,913 + 271,658 = \mathbf{379,571}$
2. **Recommended Order:** Process the 'AND' clauses in increasing order of estimated size:
   (a) Process '(kaleidoscope OR eyes)'.
   (b) Process '(tangerine OR trees)'.
   (c) Intersect (AND) the results of the first two operations.
   (d) Process '(marmalade OR skies)'.
   (e) Intersect the result from step 3 with the result from step 4.
3. **Which two terms first?** The two terms to process first are **kaleidoscope** and **eyes**, as their union forms the smallest estimated intermediate set.

Exercise 3 (Page 51)

Exercise: If the query is **friends AND romans AND (NOT countrymen)**, how could we use the freq of countrymen?

Answer: The frequency of 'countrymen' is used for **query optimization**.

- The optimizer compares the frequencies of all three terms.
- It might first intersect 'friends' and 'romans' to get an intermediate list.
- The frequency of 'countrymen' tells the optimizer how many documents will be removed by the 'AND (NOT countrymen)' operation. A small frequency means few documents are removed; a large frequency means many are removed.
- This helps decide the most efficient execution path (e.g., whether to perform the 'NOT' operation earlier or later).

Exercise 4 (Page 51)

Exercise: Extend the merge to an arbitrary Boolean query. Can we always guarantee execution in time linear in the total postings size?

Answer:

- **Extending the merge:** An arbitrary Boolean query (e.g., (A OR B) AND (C OR D)) can be processed by breaking it down into smaller sub-queries.
  1. Identify the innermost or most deeply nested operations (e.g., A OR B).
  2. Execute the appropriate merge for that operation (INTERSECT for AND, UNION for OR) to create an intermediate postings list.
  3. Use this intermediate list as an input for the next level of operations.
  4. Repeat until the entire query is resolved into a single final list.
- **Time Guarantee: No**, we cannot guarantee execution in time linear in the total postings size of the query terms. If the query contains a 'NOT' operator (e.g., 'NOT countrymen'), it requires processing a list of effectively every document in the collection that *doesn't* contain the term. This list size is proportional to the total collection size (N), which can be much larger than the sum of the individual terms' postings lists.

Exercise 5 (Page 61)

Exercise: Adapt the linear merge of postings to handle proximity queries. Can you make it work for any value of k?

Answer: Requires a **positional index**.

1. **Document-Level Merge:** Standard 'AND' merge to find documents containing both terms.
2. **Positional-Level Merge:** For each matched document, retrieve position lists for both terms. Merge these lists to check if any pair of positions $|p_1 - p_2| \leq k$.

Yes, this works for any k. The value k is just a parameter used during the positional comparison step.

Lecture 3: Index Compression

Exercise 1 (Page 7)

Exercise: Give intuitions for all the '0' entries. Why do some zero entries correspond to big deltas in other columns?

Answer: The '0' entries are in the 'positional postings' column for "Case folding" and "stemming".

- **Intuition:** Positional indexes store an entry for *every token occurrence*. Case folding and stemming merge *unique term types* in the dictionary but do not change the total number of tokens in the text. Thus, the positional index size remains unchanged (0% delta).
- **Big Deltas:**
  - **Stopwords:** removing them has little effect on dictionary size (0% delta) but a huge effect on positional postings (-38%) because stop words are extremely frequent tokens.
  - **Stemming:** has 0% effect on positional postings but -4% on non-positional postings because merging terms also merges their document lists, reducing the number of docID entries.

Exercise 2 (Page 12)

Exercise: What is the effect of including spelling errors, vs. automatically correcting spelling errors on Heaps' law?

Answer: Heaps' law: $M = kT^b$.

- **Including errors:** Increases vocabulary size (M) for the same number of tokens (T), leading to a **higher** k.
- **Correcting errors:** Merges errors into correct terms, decreasing vocabulary size (M), leading to a **lower** k.

Exercise 3 (Page 12)

Exercise: Compute vocabulary size M if: 3,000 terms in first 10,000 tokens; 30,000 terms in first 1,000,000 tokens.

Answer: Using $M = kT^b \implies \log_{10} M = \log_{10} k + b \log_{10} T$:

1. $\log_{10}(3000) = \log_{10} k + 4b \implies 3.477 = \log_{10} k + 4b$
2. $\log_{10}(30000) = \log_{10} k + 6b \implies 4.477 = \log_{10} k + 6b$

Subtracting (1) from (2) gives $1.0 = 2b \implies \mathbf{b = 0.5}$. Plugging back into (1) gives $\log_{10} k = 1.477 \implies \mathbf{k \approx 30}$.

Exercise 4 (Page 12)

Exercise: Search engine indexes 2×10^{10} pages, 200 tokens avg. What is predicted vocabulary size? (Using $k = 30, b = 0.5$)

Answer:

- Total tokens $T = (2 \times 10^{10}) \times 200 = 4 \times 10^{12}$.
- $M = 30 \times (4 \times 10^{12})^{0.5} = 30 \times 2 \times 10^{6} =$ **60** million terms.

Exercise 5 (Page 20)

Exercise: Why is/isn't 4.5 characters/word the number to use for estimating dictionary size?

Answer: It is **not** the correct number. 4.5 is the average length of *tokens* in text, heavily skewed by frequent short words (e.g., "a", "the"). The dictionary stores *unique terms*, where long, rare words are equally represented. The average unique term length is longer, around **8 characters**.

Exercise 6 (Page 25)

Exercise: If query term frequencies were non-uniform but known, how would you structure the dictionary search tree?

Answer: You would structure the tree so that the most frequently queried terms are placed near the root (top) of the tree. This ensures that the most common searches require the fewest number of comparisons, minimizing the average search time.

Exercise 7 (Page 27)

Exercise: Estimate space usage and savings with blocking for $k = 4, 8, 16$ (Baseline 7.6 MB, 400k terms).

Answer: Savings per block of k is $(2k - 3)$ bytes. Total blocks $= 400,000/k$.

- **k=4:** Savings $= 100,000 \times 5\text{B} \approx 0.5$ MB. New size: **7.1 MB**.
- **k=8:** Savings $= 50,000 \times 13\text{B} \approx 0.65$ MB. New size: **6.95 MB**.
- **k=16:** Savings $= 25,000 \times 29\text{B} \approx 0.725$ MB. New size: **6.875 MB**.

Exercise 8 (Page 27)

Exercise: Estimate impact on search performance for $k = 4, 8, 16$ compared to $k = 1$.

Answer: Avg comparisons $\approx \log_2(400,000/k) + (k+1)/2$. Baseline ($k = 1$) is ≈ 18.6.

- **k=4:** ≈ 19.1 comparisons (**1.03x slowdown**).
- **k=8:** ≈ 20.1 comparisons (**1.08x slowdown**).
- **k=16:** ≈ 23.1 comparisons (**1.24x slowdown**).